C4ISR Architectures I:
Developing a Process for C4ISR Architecture Design[1]

Alexander H. Levis and Lee W. Wagenhals
System Architectures Laboratory, C3I Center, MSN 4D2, George Mason University, Fairfax, VA 22030

<alevis>, <lwagenha> @gmu.edu
Tel: 703 993 1619; 703 993 1712;
Fax: 703 993 1708

ABSTRACT

The C4ISR Architecture Framework document issued by the Department of Defense specifies three views of an information architecture and defines a set of products that describe each view. These architecture views are to serve as the basis for C4ISR system development and acquisition. The Framework does not provide a process for architecture design. In this paper, information architectures are described in the context of Structured Analysis and then the architecture views of the Framework and the related products are interpreted in that context. A methodology for architecture design is developed that is then implemented using Structured Analysis and object orientation in two companion papers.

[1] This work was supported in part by the Office of Naval Research under Grant no. N00014-00-1-0267 and the Air Force Office for Scientific Research under Grant no. F49620-95-0134.
To Appear in Systems Engineering, Vol. 3, No. 4, Fall 2000.

1. INTRODUCTION

In a changing world, the Department of Defense has to cope with increased uncertainty about requirements, rapid changes in technology, changes in organizational structures, and a widening spectrum of missions and operations. One way to deal with these uncertainties is to be able to rapidly mix and match organizations with composite capabilities to suit a particular situation. To do this requires an unprecedented level of interoperability in information systems. To achieve this flexibility, DoD has looked to information architectures that can provide current or future descriptions of a "domain" composed of components and their interconnections, actions or activities those components perform, and rules or constraints for those activities. These architectures, while they will change over time, will change at a much slower rate than the actual systems they represent. Because of their stability, they can act as important guides to acquisition decisions as well as defining operational concepts. One domain of *information systems* that directly supports military operations is Command, Control, Communications, Computer, Intelligence, Surveillance, and Reconnaissance (C4ISR). The goal is to describe architectures using multiple views that answer operator's questions regarding the operational capability that systems built conformant to the architecture can provide. Another goal is to support the acquisition community in its efforts to acquire interoperable system. A seamless process from knowledge elicitation to architecture design and evaluation is desired.

In December 1997, the Office of the Secretary of Defense published the C4ISR Architecture Framework, Version 2.0. [C4ISR, 1997][2] In February 23 1998, the Under Secretary of Defense (Acquisition and Technology), the Assistant Secretary of Defense (Command, Control, Communications, and Intelligence) and the Joint Staff Director for Command Control Communications and Computers (C4) issued a memorandum that stated:

"We see the C4ISR Architecture Framework as a critical element of the strategic direction in the Department, and accordingly direct that all on-going and planned C4ISR or related architectures be developed in accordance with Version 2.0. Existing C4ISR architectures will be re-described in accordance with the Framework during appropriate revision cycles."

[2] The C4ISR Architecture Framework, version 2.0, document can be found in a number of DOD and non-DOD websites that require different forms of access. The System Architectures Lab maintains a copy in .pdf format at http://viking.gmu.edu/http/c4isrmay2000/index.html

The Architecture Framework provides common definitions, data, and references, and describes a set of products that comprise three views of an architecture. The Framework provides little guidance on how to produce those products. To comply with the directive, organizations must develop their own process and many of them have done so. This poses a challenge for at least two reasons. First, the three architecture views (Operational, Systems, and Technical Architecture views) and their products are new constructs that are not supported directly by existing systems engineering formalisms and tools. Second, while the products provide a great deal of information, it is not clear that, in the absence of a well thought out process, they will be mutually consistent so that they can be used to answer questions about the capabilities that systems built in conformance with the architecture will provide.

One of the issues is that the C4ISR Architecture Framework, version 2.0, makes a number of assertions (Section 1.1 of the Framework) regarding its purpose that are hard to justify in view of the community's experience in the last three years and the analysis of the constructs that has been carried out. While the application of the Framework does enable architectures to contribute to building interoperable and cost-effective military systems, and does have the potential for architectures developed by different organizations to be interrelatable and possibly comparable, it is hard to imagine how they would be integratable. The Framework does not provide guidance regarding these latter aspects.

One of the premises of the work presented in this and the two companion papers is that the derivation of an executable model of the architecture from the three views and the associated integrated dictionary provides a basis for understanding the interrelationships among the various architecture products and establishes the foundation for implementing a process for assessing and comparing architectures. The paper does not advocate that the creation of an executable model be mandated as part of the Architecture Framework. Rather, it argues that given the particular form of the Framework products and the absence of well established processes for developing a set of consistent and coherent products, architects may be well advised to create an executable model as a means analyzing the characteristics of their architecture and ascertain whether behavioral requirements are met.

This article describes the role of the architect, establishes the criteria for determining the data needed in an information architecture to support the role of the architect, examines the C4ISR

Framework products, and provides a mapping from the Structured Analysis products to the Framework products. This mapping forms the basis for two alternative approaches documented in the companion papers, one based on Structured Analysis [Wagenhals et al., 00] and one based on Object-Orientation [Bienvenu et al., 00].

To address these issues, the remainder of this article is divided into five sections. Section 2 describes the fundamentals of architectures and the role of the architect. Section 3 presents a generic process for creating an architecture of an information system that is capable of describing behavioral and performance aspects of the architecture to the customer. Section 4 describes the C4ISR Architecture Framework construct, including a brief description of the three architecture views and their products. Section 5 shows the mapping from the Structured Analysis products to the Framework products; this mapping sets the basis for the alternative approaches.

2. ON ARCHITECTURES

2.1 Architects and Architectures

The concept of architectures and the business of creating them have been around for millennia. Indeed, the Greek word αρχιτεκτων (architecton) means master builder or master mason. The term describes one who designs and builds structures whose form and function are both appealing and useful. Today, when we think of architectures, we think of buildings and monuments that are the creation of named architects. The main contribution of the architect is the conceptualization and design of a unique structure to meet the client's needs. The architect has the special role of eliciting and converting the needs and desires of the customer that commissions him into a design that will be especially satisfying to that customer.

In a modern context, we think of hiring an architect when we want to create a structure that is both unprecedented and complex, particularly if our needs are initially ill structured. [Rechtin, 91, 92] An example is the building of a custom house. Assuming that one is interested in more than just a house whose plans can be downloaded from the internet or a "standard" house offered by a national or local building company, one may decide to hire an architect to create the description of the house. The architect will elicit needs and desires from the customer and, using a combination of engineering and art, will create a series of models (drawings, scale replicas, 3D visualizations) of the house. The models are abstract at first and become more specific as the

customer provides feedback to the architect. Once an architecture that is satisfactory to the customer has been created, the architectural models are converted into a set of blueprints that are provided to a general contractor who is responsible for building the house in accordance with the blueprints that have been derived from the architectural drawings. Note also that the actual construction must be compliant with the building codes that apply to that type of structure in that particular locality.

From this example and by analogy, it is possible to derive several characteristics of system architects and the architectures they produce. The first is that an architect is needed only if the system is unprecedented and complex. [Rechtin, 91, 92] If satisfactory, working systems have already been built and the designs exist, there is no need for an architect. An architect is not needed if the system is very simple and can be constructed directly by a contractor. The architect's responsibility is different from the general contractor's. The architect is driven by the special needs of the customer and tends to develop the architecture in a top-down manner. [Rechtin, 91, 92] Indeed, the task of the architect is to elicit those needs and to produce a description that can demonstrate to the customer that the system to be produced in conformance with the architecture will satisfy the customer's wants and needs. This means that the architecture does not include the details of the final system designs.

A key issue in the above description is where the architect's work in describing the architecture ends and the system design begins. The blueprints are usually thought as the representation of a design – the implementation of an architecture, not as a description of an architecture. In the Framework, the architecture description is referred to as a blueprint that then enters the design, development, and acquisition process. The problem of demarcation between architecture and design (never an easy one to determine) is especially daunting in the case of C4ISR systems where any future architecture will be populated in the design phase by existing systems, the so-called legacy systems. So, while the architecture definition process is thought of as being top-down, the presence of so many legacy systems introduces constraints in the design of the architecture that are best addressed in a bottom-up approach. This particular issue is one of the drivers for the approach presented in this paper and forms the basis for some of the conclusions drawn.

The architect develops and presents the architecture as a set of abstract views or models. The abstraction occurs in two dimensions, *generalization* and *composition*. The architect begins with a very *general* description of the system. Based on discussions with the customer, the architect arranges and *specializes* these components to suit the customer's needs and desires. Note that the number of components in the architecture and, thus, its complexity does not need to increase substantially in this specialization process. The number of components does increase as more *detail* is incorporated in the architecture. Adding detail is generally accomplished by *decomposing* the basic components of the architecture into their constituent parts. Decomposition can significantly increase the number of components of the architecture and thus its complexity. Because of this, the architect should use decomposition only as necessary to address the questions and concerns of the customer. The customer and the architect assume that these components will work properly because they will be constructed and installed in accordance with established codes and guidelines. The actual system design and implementation will involve the specialization of the architecture and the addition of all of the details of the design so that the system can be manufactured. This specialization process is the task of the general contractor or system engineer. The actual system will be the most specialized version of the architecture. There can be many ways of specializing a single architecture into actual systems. The selection of the actual design can be determined by cost and technology factors. Thus, an architecture can be a valid description of a way of satisfying a customer's need over a long period, even as the specific techniques for implementing the architecture change.

The concepts and characteristics applicable to architecting buildings also apply to architecting information systems. One thinks of an architecture as being implemented by many diverse interacting systems. The architect needs to be knowledgeable not only about the individual systems, but also about the interrelationships among them. Furthermore, the architect must use creativity and vision because of the unprecedented and complex nature of the design and the lack of an initial clear definition of the needs and requirements for the system [Levis, 99]. It is important that the architect be able to show and discuss with the customer what properties the architecture will have. This means that the architectural models must be capable of providing insight into the logical and behavioral aspects of the architecture and the performance aspects of systems that are conformant to the architecture. This important criterion influences the process and the techniques for developing an architecture of a system.

Systems Architecting [Rechtin, 91, 92; Rechtin and Maier, 96] is part of the system engineering process and relies on many of the methodologies that have been developed over time. The architect has many tools and techniques available to describe the architecture. Two major paradigms that are appropriate are the traditional Structured Analysis and Design Technique (SADT) and the Object-Oriented approach that originated with software systems. Both offer advantages and both are discussed at length in this and the two companion articles [Wagenhals et al., 00; Bienvenu et al., 00]. Both approaches can fall short of the requirement of being able to convey the logical, behavioral, and performance properties of the architecture. This is because both approaches rely on static pictures, diagrams, and textual descriptions to define the architecture. However, an architecture is instantiated with dynamic systems that interact with their environment over time. To fully describe and understand the dynamic aspects of the system requires an executable model.

2.2 System Architecting using Structured Analysis

In the basic systems engineering approach using Structured Analysis, an architecture is composed of two constructs: the *Functional Architecture (view)* and the *Physical Architecture (view)*[3]. The *Technical Architecture view* in the Framework, while not part of the Structured Analysis approach, corresponds, in broad terms, to the building code that any architect must take into consideration. The technical architecture view is included in the discussion because of its relevance to the DOD goal of acquiring interoperable systems. The term view is used to emphasize that there is a single architecture of the system. The complete representation of that architecture requires different views. These views can be categorized by their perspective.

In defining an architecture, several perspectives need to be described. First, since the architecture will be created so that the systems that populate it will perform some useful function, we need to describe the process or activities that need to take place in order for the systems to accomplish their purpose. In an information system, the processes receive and transform data that "flow" between them. These processes or activities follow rules that

[3] In the systems engineering literature, the term system architecture has often been used to denote the physical architecture. This has been the source of much confusion because the term system architecture used in the C4ISR Architecture Framework denotes a different construct.

determine the conditions under which they occur and the type of outputs they produce. In addition, the processes should occur in some order based on the initial conditions of the system, and several processes may occur concurrently and asynchronously. In addition to the processes or activities, it is necessary to describe the components that will implement the design: the hardware, software, personnel, and facilities that will comprise the C4ISR system and perform the processes.

This fundamental notion leads to the definition of the two basic architectural constructs in Structured Analysis. A *Functional Architecture* is a set of activities or functions, arranged in a specified partial order that, when activated, achieves a set of requirements. Similarly, a *Physical Architecture* is a representation of the physical resources, expressed as nodes, that constitute the system and their connectivity, expressed in the form of links. Both definitions should be interpreted broadly to cover a wide range of applications; furthermore, each may require multiple representations or views to describe all aspects.

Before even attempting to develop these representations, the operational concept must be defined. This is the first step in the architecture development process. An *Operational Concept* is a concise statement that describes how the goal will be met. There are no analytical procedures for deriving an operational concept for complex, unprecedented systems. On the contrary, given a set of goals, experience, and expertise, humans invent operational concepts. It has often been stated [Rechtin, 91] that the development of an architecture is both an art and a science. The conceptualization of an operational concept falls clearly on the art side. A good operational concept is based on a simple idea of how the over-riding goal is to be met. For example, "centralized decision making and distributed execution" represents a very abstract operational concept that lends itself to many possible implementations, while an operational concept such as the "client-server" one is much more limiting. As the architecture development process unfolds, it becomes necessary to elaborate on the operational concept and make it more specific. The clear definition and understanding of the operational concept is central to the development of compatible functional and physical architectures.

Analogous to the close relationship between the operational concept and the functional architecture (to the extent that often a graphical description of the operational concept is improperly presented as the functional architecture,) is the relationship between the technical

architecture and the physical one. A *Technical Architecture (TA) view* is a minimal set of rules governing the arrangement, interaction and interdependence of the parts or elements whose purpose is to ensure that a conformant system satisfies a specified set of requirements.[C4ISR, 97] It provides the framework upon which engineering specifications can be derived, guiding the implementation of the system. It can be compared to that part of the building code that provides guidance for the new building to be able to connect to the existing infrastructure of its planned location by characterizing the attributes of that infrastructure, as well as the exceptions to the code.

It is the Technical Architecture that provides the connection between the abstract descriptions of Physical and Functional Architecture views and the implementation of the detailed system design. When architects define the Technical Architecture view, they are providing guidance on the further specialization and decomposition of the components of the Physical and Functional Architecture that will be accomplished in the detailed engineering design of the system.

All of these representations of the architecture, even when they describe the dynamic behavior of the architecture, are static. They are inadequate for analysis of the behavior and performance aspects of the architecture. In the next section, the details of the models used in these representations are described. They contain a great deal of information but, in general, they are ill suited to answering the main concerns of the customer. In order to analyze the behavior and performance of the architecture and address the concerns of the customer, an executable model is derived from them. After all, the systems to be designed are dynamic systems. An executable model is a dynamic model; it can be used to analyze the properties of the architecture and it can be used to carry out simulations. However, it also serves in a subtler, but very important role. It becomes the litmus test by which one can determine whether the description of the system architecture, as given by a set of static representations or models, is sufficient with respect to the set of questions to be asked of the architecture when these questions involve the dynamic behavior of the C4ISR system[4]. Indeed, the methodologies, whether Structured Analysis based or Object-Oriented based, become rigorous when an executable model is derived and *the condition is imposed that all information contained in the executable model must be traced back to one or more of the static views.*

[4] Further discussion on this issue appears in Section 5.

The architecture development process can be characterized as consisting of three phases: the *Analysis* phase in which the static representations of the Functional and Physical Architecture views are obtained using the operational concept to drive the process and the Technical Architecture view to guide it, the *Synthesis* phase in which these static constructs are used, together with descriptions of the dynamic behavior of the architecture (often referred to as the *Dynamics* model), to obtain the executable model of the architecture, and the *Evaluation* phase in which measures of performance (MOPs) and measures of effectiveness (MOEs) are obtained. This three phase process is shown schematically in Fig. 1.

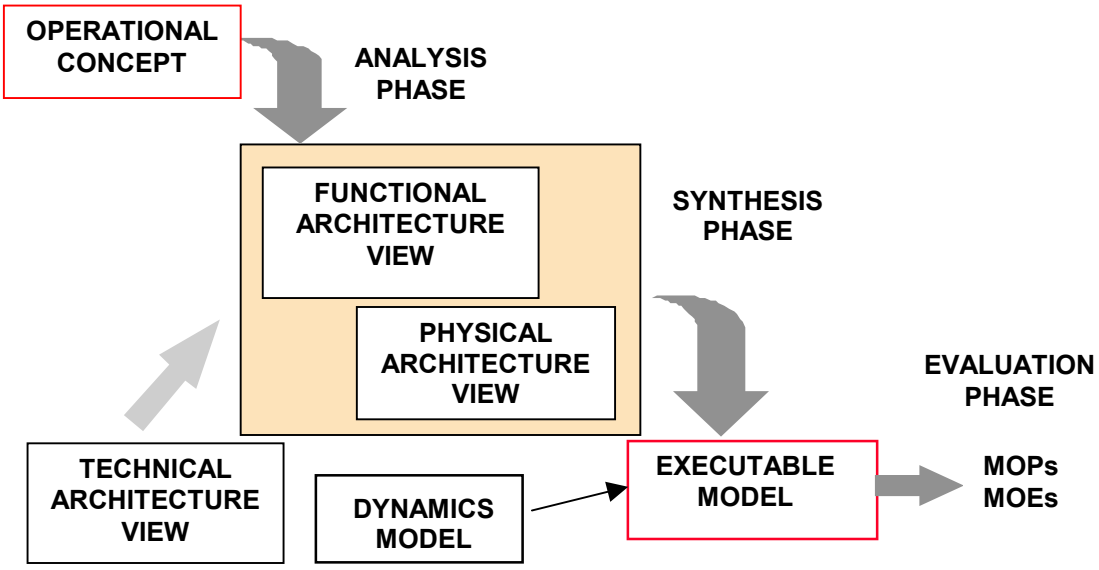

Figure 1. The Three Phases of Architecture Development

A key use of the executable model is to allow the architect to shift the locus of discourse with the customer or user from the architecture views to the behavior and performance that the architecture enables. This concept is shown in Fig. 2. The figure points out that the static descriptions of the architecture are a means to an end; the end is to provide the C4ISR system users with capabilities that support their missions. Note that while a number of questions can be resolved with the use of only of few of the static products, experience with a number of DOD organizations has shown that communicating with users via these constructs is not an effective process, even though these constructs may contain within them all the data needed to provide the answer.

The next section describes the first of the two paradigms for the architecture development process: Structured Analysis and its tools. The reason is that the C4ISR Architecture Framework, as currently described, has a strong Structured Analysis flavor. While Object-Orientation is not excluded explicitly, the language, terminology, and examples used all are based on Structured Analysis. However, while the two approaches are conceptually different, many of the same tools can be used to construct the various representations that comprise each approach. Furthermore, once the executable model is obtained, whether through Structured Analysis or Object-Orientation, the evaluation phase is the same. After discussing the evaluation phase, we will show in Section 5 how to map the information contained in the Structure Analysis products into the C4ISR Architecture Framework Version 2 essential products. It will be clear that these products can be derived from a well defined description of the architecture regardless of the approach used to create the architecture; two equivalent but not identical sets of supporting products can also be derived.

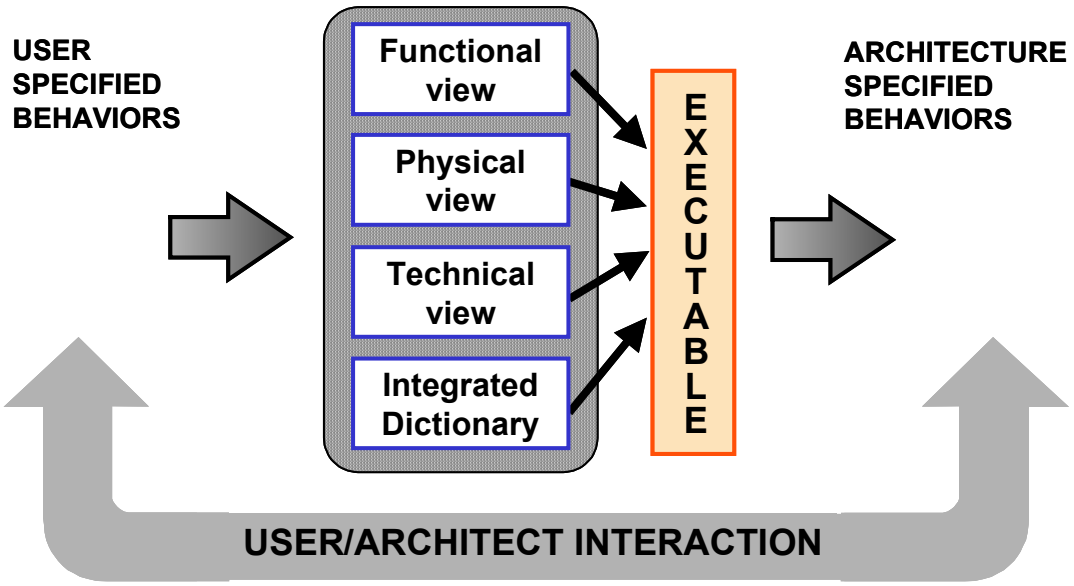

Figure 2. Behavior as the Locus of Discourse

3. STRUCTURED ANALYSIS APPROACH

The Structured Analysis approach has its roots in the Structured Analysis and Design Technique (SADT) that originated in the 50's [Marca and McGowan, 87] and encompasses Structured Design [Yourdon and Constantine, 75], Structured Development [Ward and Mellor, 86], the Structured Analysis approach of DeMarco [79], Structured Systems Analysis [Gane and Sarson, 78], and the many variants that have appeared since then, often embodied in software packages for computer-aided requirements generation and analysis. This approach can be characterized as a process-oriented one [Solvberg and Kung, 93] in that it considers as the starting point the functions or activities that the system must perform. A second characterizing feature is the use of functional decompositions and the resulting hierarchically structured diagrams. However, to obtain a specification of the architecture that allows the derivation of the executable model, in addition to the process or activity model, a data model, a rule model, and a dynamics model are required. Each one of these models contains inter-related aspects of the architecture description. For example, in the case of an information system, the activities or processes receive data as input, transform it, and produce data as output. The associated data model describes the relationships between these same data elements. The activities take place when some conditions are satisfied. These conditions are expressed as rules associated with the activities. But for the rules to be evaluated, they require data that must be available at that particular activity with which the rule is associated; the output of the rule also consists of data that control the execution of the process. Furthermore, given that the architecture is for a dynamic system, the states of the system need to be defined and the transitions between states identified to describe its dynamic behavior. State transition diagrams are but one way of representing this information. Underlying these four models is a data dictionary or, more properly, an integrated system dictionary, in which all data elements, activities, rules, and flows are defined. The construct that emerges from this description is that a set of inter-related views, or models, is needed to describe an architecture using the structured analysis approach.

In an ideal world, a tool would exist that would support all these models of an architecture and generate a consistent data dictionary. While the four types of models exist in many forms and software tools for their generation are available, they have been developed independently from a different starting point: it is possible to approach the problem by starting with a data model (data oriented approach) or with a rule model (rule oriented approach). At this time, the architect must use a suite of tools and, cognizant of the inter-relationships among the four models and the

features of the tools chosen to depict them, work across models to make the various views consistent and coherent. Moreover, he must obtain a single, integrated system dictionary from the individual dictionaries generated by the various tools.

The Activity model, the Data model, the Rule model and the supporting Integrated System Dictionary, taken together, constitute the Functional Architecture view of the system (Fig. 3). The term Functional Architecture has been used to describe a range of representations -- from a simple activity model to the set of models defined here. What a Functional Architecture does not contain is the specification of the physical resources that will be used to implement the functions or the structure of the human organization that is supported by the information system. These descriptions are contained in the Physical architecture.

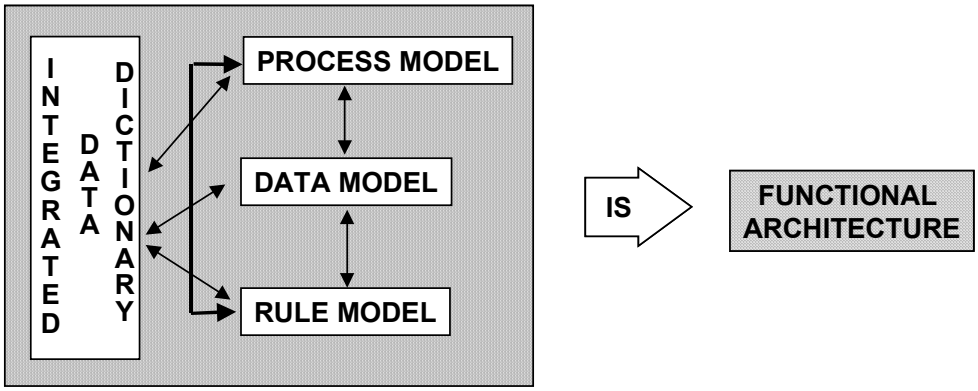

Figure 3. Components of the Functional Architecture

3.1 Functional Decomposition and Activity Model

In a process oriented approach such as structured analysis, the architecture development process can start with a very abstract operational concept. As the analysis evolves, the operational concept becomes more specific. The operational concept is often described pictorially with an associated narrative that explains how the operation is to take place. Cartoons and Clip Art are often used in the graphical representation. This depiction is often inappropriately referred to as an architecture. It is not. It is equivalent to a sketch that an architect

may make of a house, how it sits on the land and where the main functional areas are. It is not the model of the house itself or the schematics that would allow someone to build this house.

Given an operational concept at some level of abstraction, the first step in the development of the functional architecture is the *Functional Decomposition.* Starting with a verb or verb phrase that articulates the function of the system, a first level decomposition is done into functions that *are part of* the top level function. These first level functions are mutually exclusive and could be totally exhaustive. Each of these functions can be decomposed further into level two functions that are parts of it, and so forth. This decomposition can be shown in outline form or graphically as a tree structure. In various approaches, specific names are given to the decomposition levels: for example "mission -- function -- task" is one set of labels (Fig. 4). The decomposition is carried out to as many levels as is necessary, always guided by the operational concept. However, keeping the levels as few as possible is recommended for two reasons: each additional level increases substantially the complexity of the problem (and may not be supportable by the other parts of the architecture process) and because a multilevel decomposition may introduce implicitly a physical architecture - the way the functions are partitioned may specify prematurely implementation solutions. One useful rule is to decompose until each function can be assigned to a single physical resource. To achieve that, it is implied that the physical architecture is available. This is usually not the case; *the two should be developed in parallel with much interaction between them*. Indeed, the functional decomposition is an iterative process and should be done with care; it is difficult to go back to the higher levels and make changes to them -- the lower levels and everything related to them will have to be re-examined.

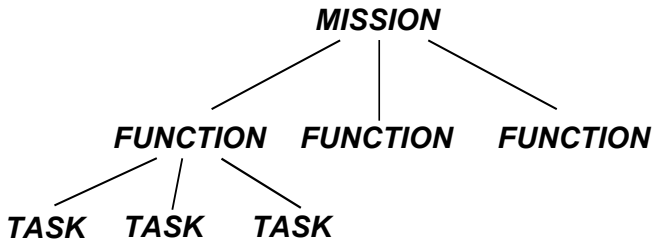

Figure 4. The Tree Structure of the Functional Decomposition

There are two primary methods in wide use for the representation of an Activity Model. The first, IDEF0[5], has systems engineering roots, while the second, Data Flow Diagrams, has its roots

[5] IDEF0 stands for ICAM (Integrated Computer Aided Manufacturing) Definition Language 0.

in software systems engineering. The National Institute of Standards and Technology (NIST) has published Draft Federal Information Processing Standard #183 for IDEF0; this is not the case for DFD. There are various variants and extensions of the Data Flow diagramming approach. Each of the approaches has advantages and disadvantages; choosing one of them depends on the features of the problem to be addressed. For the history of IDEF0, see Marca and McGowan [88] and for Data Flow Diagrams, see Yourdon [89] or Rumbaugh et al. [91]. IDEF0 is described briefly in this section.

IDEF0 is a subset of the Structured Analysis and Design Technique (SADT). It is a modeling language for developing structured graphical representations of the activities or functions of a system. It has been designed to describe and aid in understanding complex systems. It is a static representation, designed to address a specific question from a single point of view. It has two graphical elements: a box, which represents an activity, and a directed arc that represents the conveyance of data or objects related to the activity. A distinguishing characteristic of IDEF0 is that the sides of the activity box have a standard meaning, as shown in Fig. 5. Arcs entering the left side of the activity box are inputs, controls enter the top side, and mechanisms or resources used to perform the activity enter the bottom side. Arcs leaving the right side are outputs -- the data or objects generated by the activity. When IDEF0 is used to represent the activity model in a Functional Architecture, mechanisms are not needed; they are part of the Physical Architecture.

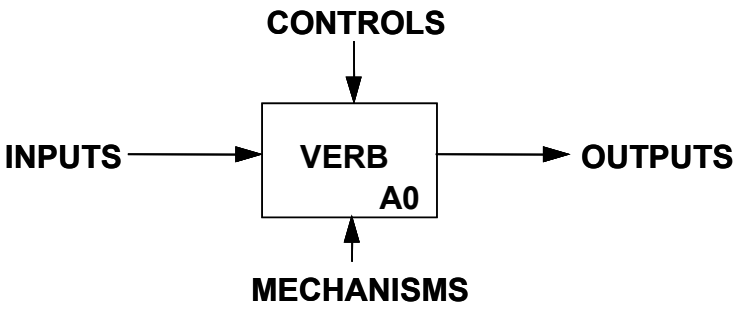

Figure 5. Box and Arrow Semantics in IDEF0

Verbs or verb phrases are inscribed in the activity boxes to define the function represented. Similarly, arc inscriptions are used to identify the data or objects represented by the arcs. There are detailed rules for handling the branching and the joining of the arcs. [FIPS 183]

A key feature of IDEF0 is that it supports hierarchical decomposition. At the highest level, the A-0 level, there is a single activity that contains the root verb of the functional decomposition. This is called the context diagram and also includes a statement of the purpose of the model and the point of view taken. The next level down, the A0 level, contains the first level decomposition of the system function and the interrelationships between these functions. It is a single page. Each one of the activity boxes on the A0 page can be further decomposed into the A1, A2, A3, ... page, respectively. A typical IDEF0 diagram of the first two levels -- A-0 and A0 -- is shown in Fig. 6. There are two inputs, one control, and three outputs.

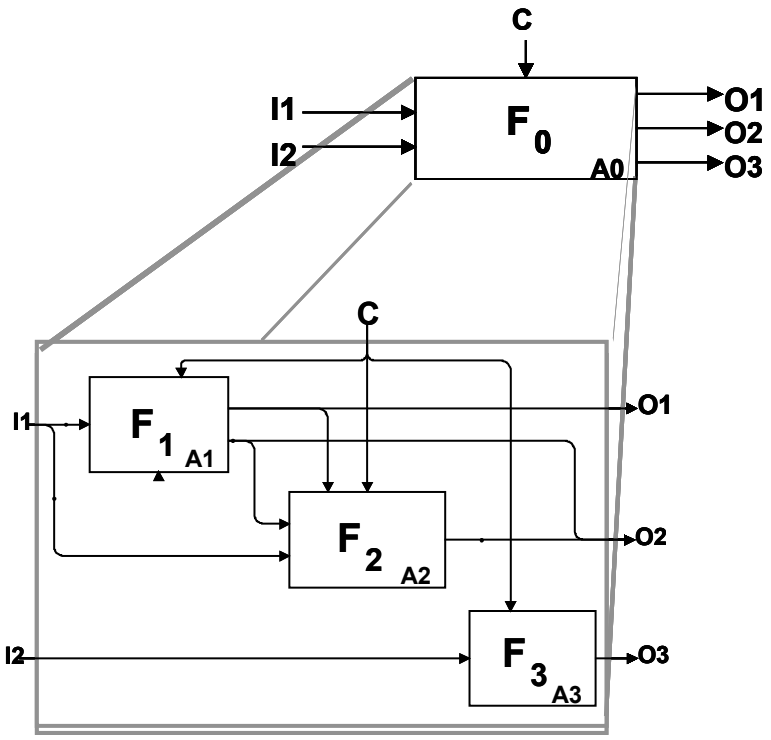

Figure 6. First Two Levels of an IDEF0 Model

Associated with IDEF0 is a data dictionary which includes the definitions and descriptions of the activities, listing and description of the inputs, controls, and outputs, and, if entered, a set of activation rules of the form "preconditions --> postconditions." These are the rules that indicate the conditions under which the associated function can be carried out. In using IDEF0 to represent an information system (a class of systems not well suited to an IDEF0 representation when one considers the origins of language) it is appropriate to characterize or type the data elements contained in the arcs. This is better done, however, in the data model.

3.2 Data Model

The arcs in the activity model of an information architecture represent data or objects. The purpose of a data model is to analyze the data structures and their relationships independently of the processing that takes place, already depicted in the activity model. There are two main approaches with associated tools for data modeling: IDEF1x and Entity-Relationship (E-R) diagrams. Both approaches are used widely. The National Institute of Standards and Technology has published Draft Federal Information Processing Standard #184 [FIPS 184] in which IDEF1x is specified. There are many books that describe E-R diagrams: Sanders [95], Yourdon [89], and McLeod [94].

IDEF1x (IDEF1 extended) is a modeling language for representing the structure and semantics of the information in a system. The elements of IDEF1x are the entities, their relationships or associations, and the attributes or keys. An IDEF1x model is comprised of one or more views, definitions of the entities, and the domains of the attributes used in the views.

An entity is the representation of a set of real or abstract objects that share the same characteristics and can participate in the same relationships. An individual member of the set is called an entity instance. An entity is depicted by a box; it has a unique name and a unique identifier. If an instance of an entity is identifiable with reference to its relationship to other entities it is called identifier dependent. A slightly different form of the box is used to distinguish identifier independent and dependent entities. The box depicting the entity instance is divided into two parts: the top part contains the primary key attributes; the lower one the non-primary key attributes. Every attribute must have a name (expressed as a noun or noun phrase) that is unique among all attributes across the entities in the model. The attributes take values from their specified domains – this automatically establishes the type of the attribute. This formalism is shown in Fig. 7.

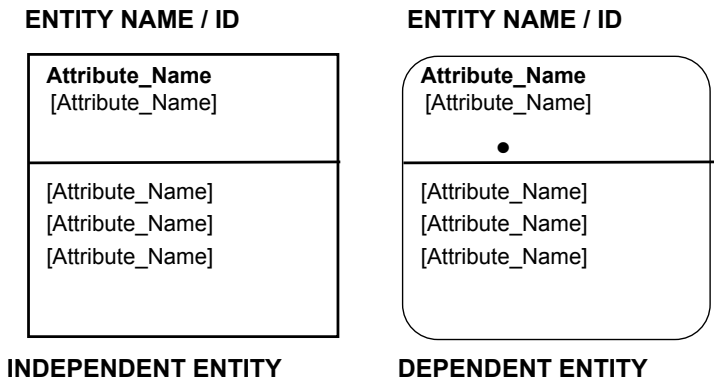

Figure 7. Independent and Dependent entities with Key and Non-Key attributes

Relationships between entities are depicted in the form of lines that connect entities; a verb or verb phrase is placed beside the relationship line. The connection relationship is directed -- it establishes a parent-child association -- and has cardinality. Special symbols are used to at the ends of the lines to indicate the cardinality. The relationships can be classified into types such as identifying or non-identifying, specific and non-specific, and categorization relationships. The latter, for example, is a generalization/specialization relationship in which an attribute of the generic entity is used as the discriminator for the categories. A simple example is shown in Fig. 8 in which three types of vehicles are defined as categories of vehicles.

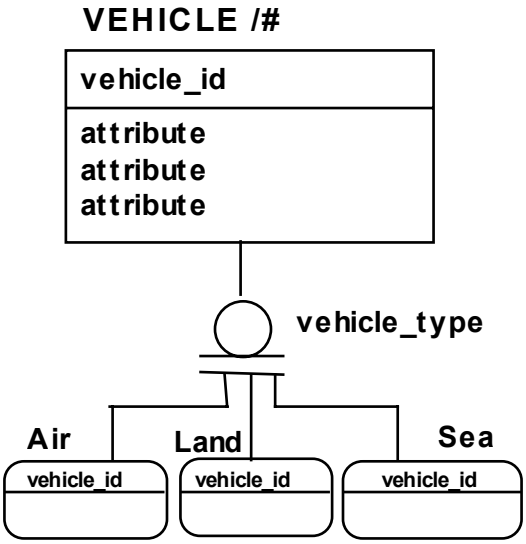

Figure 8. Example of generic and category entity specification using a discriminator

3.3 Rule Model

In a rule oriented model, knowledge about the behavior of the architecture is represented by a set of assertions that describe what is to be done when a set of conditions evaluates as true. These assertions, or rules, apply to specific functions defined in the activity model and are formulated as relationships among data elements. There are several specification methods that are used depending on the application. They include Decision Trees, Decision Tables, Structured English, and Mathematical Logic. Each one has advantages and disadvantages; the choice often depends on the way that knowledge about rules has been elicited and on the complexity of the rules themselves.

A Decision Tree is most appropriate when each rule has as a consequent a single action, the execution of an activity. A Decision Tree has a single root that represents the first decision. Subsequent decisions are depicted as branches and leaves of the tree. Each node of the tree represents a decision, while the leaves indicate the resulting actions. A Decision Table is more useful when a set of conditions that evaluates as true results in multiple actions. The table is partitioned in four sections as shown in Fig. 9. The Condition stub contains the list of the decision variables, while the Action stub contains the list of actions. The Condition matrix consists of columns with each column depicting a combination of conditions. The columns of the Action matrix show which actions are to take place when the conditions in the corresponding condition column evaluate as true.

Both Decision Trees and Decision Tables show only selection or decision constructs; they do not show sequencing or repetition and iteration. Structured English shows all three control structures. In Structured English, the rules are expressed using nested patterns of the form “if-then-else-so.” The action that results when the conditions evaluate as true is expressed in the form of a command: then do this.

CONDITION STUB	CONDITION MATRIX
ACTION STUB	ACTION MATRIX

Figure 9. The four parts of a Decision Table

Of course, mathematical logic can be used (whether Symbolic logic or Predicate logic) to represent the set of rules. This is a very general representation that allows for the modeling of very complex rules.

3.4 Dynamics Model

The fourth type of model that is needed is one that characterizes the dynamic behavior of the architecture. This is not an executable model, but one that shows the transition of the system state as a result of events that take place. The state of a system can be defined as all the information that is needed at some time t_o so that knowledge of the system and its inputs from that time on determines the outputs. The state space is the set of all possible values that the state can take.

There is a wide variety of tools for depicting the dynamics, with some tools being more formal than others: state transition diagrams, state charts, event traces, key threads, etc. Each one serves a particular purpose and has unique advantages.

A State Transition diagram is a representation of a sequence of transitions from one state to another -- as a result of the occurrence of a set of events -- when starting from a particular initial state or condition. The states are represented by nodes (e.g., a box) while the transitions are shown as directed arcs. The event that causes the transition is shown as an arc annotation, while the name of the state is inscribed in the node symbol. If an action is associated with the change of state, then this is shown on the connecting arc, next to the event. The rules that enable the event to effect the transition from one state to another can be shown also on the connecting arc in brackets. An example is shown in Fig. 10.

Figure 10. A transition from State 1 to 2.

Note that the Dynamics model is not an executable model. It characterizes in a static manner aspects of the dynamic behavior of the model. Furthermore, since a State Transition diagram represents the transitions from an initial condition and a sequence of events, it follows that to characterize a system's behavior, many such diagrams are needed.

3.5 Integrated Dictionary and Model Concordance

Underlying all these four models is the Integrated System Dictionary. Since the individual models contain overlapping information, it becomes necessary to integrate the dictionaries developed for each one of them. Such a dictionary must contain descriptions of all the functions or activities including what inputs they require and what outputs they produce. These functions appear in the activity model (IDEF0), the Rule model (as actions), and the State Transition diagrams. The rules, in turn, are associated with activities; they specify the conditions that must hold for the activity to take place. For the conditions to be evaluated, the corresponding data must be available at the specific activity -- there must be an input or control in the IDEF0 diagram that makes that data available to the corresponding activity. Of course, the system dictionary contains definitions of all the data elements as well as the data flows that appear in the activity model.

The process of developing a consistent and comprehensive dictionary provides the best opportunity for ensuring concordance among the four models. Since each model has different roots and was developed to serve a different purpose, together they do not constitute a well integrated set. Rather, they can be seen as a suite of tools that collectively contain sufficient information to specify the architecture. The inter-relationships among models are complex. For example, rules should be associated with the functions at the leaves of the functional

decomposition tree. This implies that, if changes are made in the IDEF0 diagram, then the rule model should be examined to determine whether rules should be reallocated and whether they need to be restructured to reflect the availability of data in the revised activity model. A further implication is that the four models cannot be developed in sequence. Rather, the development of all four should be planned at the beginning with ample opportunity provided for iteration, because if changes are made in one, they need to be reflected in the other models.

Once concordance of these models has been achieved, it is possible to construct an executable model. Since the physical architecture has not been constructed yet, the executable model can only be used to address logical and behavioral issues, but not performance issues.

3.6 Conversion to the Executable Model

Information Systems are dynamic in nature. Events occur that trigger the execution of functions and many functions can be executed concurrently. An executable model of an information architecture enables the architect to analyze the dynamic behavior of the architecture, identify logical and behavioral errors not easily seen in the static descriptions, and demonstrate to the customer or user the capabilities that the architecture enables. There exist some graphical modeling approaches that allow a dynamic representation of a discrete event system. Colored Petri Nets, Finite State Machines, and Behavior Diagrams are examples of such approaches. They can be used directly to model a discrete event dynamical system representation of an *information architecture*. The problem, however, is that they are much more complex than the four models already described and require substantial experience and expertise to ensure that they include all relevant system aspects. The solution is, therefore, to derive the executable model from the static representations. A methodology has been developed that allows the derivation of a Colored Petri Net model of an architecture that can be traced back to the four models. Note that the discrete event system representation is not suitable for representing many physical systems that best characterized by time driven processes as described by differential or difference equations. The logical and decision processes of a C4ISR system supporting the Command and Control process are an appropriate domain for discrete event modeling since the hardware consists primarily of computers and the overall system is software intensive.

Colored Petri Nets [Jensen, 92] are a generalization of Petri Nets. The latter are bipartite directed multigraphs that are executable. In Petri Nets, two types of nodes are defined: Places

and Transitions. The arcs that join two nodes are directed; furthermore, arcs can connect only nodes of different types. Directed arcs connecting places to transitions establish the inputs to that transition, while arcs connecting transitions to places establish the outputs. The arcs can have inscriptions that define the degree of multiplicity of that arc. An illustrative Petri Net is shown in Fig. 11.

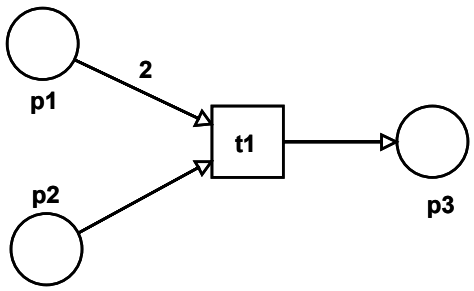

Figure 11. A Petri Net

In order for the Petri Net to execute and be a dynamical model, another element needs to be introduced. This is the token. A token is an indistinguishable marker that resides in places. The distribution of tokens in the places of a Petri Net is called a marking and defines the state of the system or net. Markings enable transitions that can then fire. The execution rule is as follows. A transition is enabled if every one of its input places has at least as many tokens as the multiplicity inscribed on the arc connecting the place to the transition. An enabled transition can fire. When it fires, the tokens used to satisfy the enablement condition are removed from the net; new indistinguishable tokens are generated in the output places of the transition. The number of tokens generated depends on the multiplicity of the outgoing arcs. Fig. 12 shows an initial marking for the net of Fig. 11 that enables the transition and the results of the transition firing.

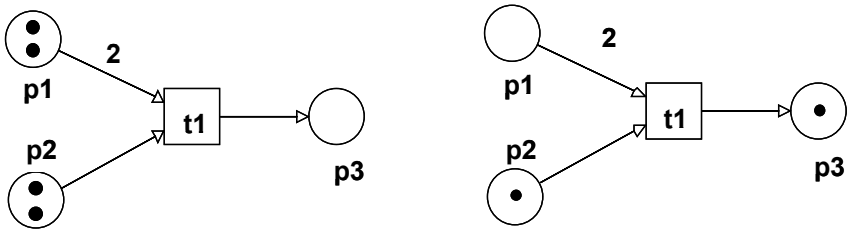

Figure 12. Enablement and Firing of Transition

In Colored Petri Nets, the tokens are distinguishable; they are characterized by their color: an attribute vector is associated with each token. The assignment of values to the attributes from their respective domains specifies the color of the token. Color sets are associated with places; they specify which token can reside in that place. Instead of a simple multiplicity number inscription that was allowed on the arcs, now complex enablement conditions can be specified. Each input arc inscription specifies the number and type of tokens that need to be in the place for the transition to be enabled. The output arc inscriptions indicate what tokens will be generated in an output place when the transitions fires. Furthermore, guard functions associated with transitions are allowed. These guard functions specify additional conditions that must be satisfied, i.e., in addition to those inscribed on the arcs, for a transition to be enabled. Code segments can be associated with transitions. These code segments can represent the function modeled by the transition and complement the output arc inscriptions.

Each Colored Petri Net model has associated with it a Global Declaration node that contains the definitions of all Color Sets and their associated domains and the definition of variables. It becomes apparent then that much of the data in the data dictionary appears in the global declaration node of the Colored Petri Net model.

The use of Colored Petri Nets to develop an executable model from the Structured Analysis models can be described as follows. One starts with the activity model. Each IDEF0 activity is converted into a transition; each IDEF0 arrow connecting two boxes is replaced by an arc-place-arc combination, (Fig. 13), and the label of the IDEF0 arc becomes the color set associated with the place. All these derived names of color sets are gathered in the Global Declaration Node. From this point on, a substantial modeling effort is required to make the Colored Petri Net model a dynamic representation of the system. The information contained in the data model is used to specify the color sets and their respective domains, while the rules in the rule model result in arc inscriptions, guard functions, and code segments.

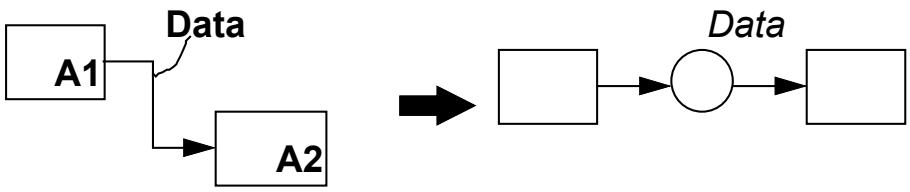

Figure 13. From IDEF0 to a Colored Petri Net representation

The process of deriving the executable model invariably leads to some revision of the static models. It is most important that discipline be exercised so that any change introduced at the executable model level is reflected back in the static models. Only this way a documented and easily reviewed representation of the architecture can be maintained.

The executable model becomes the integrator of all the information; its ability to execute tests some of the logic of the model. Given the Colored Petri Net model, a number of analytical tools from Petri Net theory can be used to evaluate the structure of the model, e.g., determine the presence of deadlocks, or obtain its occurrence graph. The occurrence graph represents a generalization of the State Transition Diagram model. By obtaining the occurrence graph of the Petri Net model, which depicts the sequence of states that can be reached from an initial marking (state) with feasible firing sequences, one has obtained a representation of a set of State Transition Diagrams. This can be thought as a first step in the validation of the model at the behavioral level. Of course, the model can be executed to check its logical consistency, that is, to check whether the functions are executed in the appropriate sequence and that the data needed by each function are appropriately provided. Performance measures cannot be obtained until the physical architecture is introduced; it provides the information needed to compute performance measures.

Since Colored Petri Nets with their dense annotation are not very easily accessible to the information system users, all the information gathered in the design and exploitation of the executable model need to be brought back into the static models. This annotated and validated representation now constitutes a sound basis for system development.

3.7 The Physical Architecture

To complete the Analysis phase of the procedure (Fig. 14), the Physical Architecture needs to be developed. There is no standardized way to represent the physical systems - existing ones as well as planned ones that will be used to implement the architecture. They range from wiring diagrams of systems to block diagram representations to node models to organization charts. While there is not much difficulty in describing in a precise manner physical subsystems using the terminology and notation of the particular domain (communication systems, computers,

displays, data bases), a problem arises on how to depict the human organization that is an integral part of the information system. The humans in the organization can not be thought simply as users; they are active participants in the workings of the information system and their organizational structure that includes task allocations, authority, responsibility, reporting requirements, etc., must be taken into account and be a part of the physical model description. This is an issue of current research, since traditional organizational models do not address explicitly the need to include the human organization as part of the physical system description.

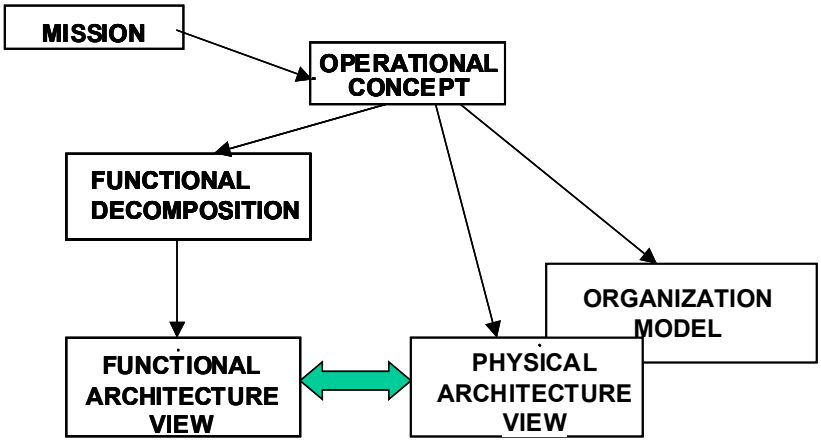

Figure 14. The Analysis Phase

Once the physical architecture is available, then an Executable Model that can be used for performance evaluation can be obtained. The Synthesis phase is described in Fig. 15. In this phase, the information and knowledge generated in the Analysis phase are synthesized into the Executable Model. As Fig. 15 shows, this can be viewed as a two stage process. First, an executable model can be created based on the Functional Architecture View. This Executable model can reveal logical and behavioral properties of the architecture that can be the focus of discourse with the customer or stakeholders. Once satisfactory behavior has been demonstrated, the information contained in the Physical Architecture View can be incorporated in the Executable Model for performance evaluation. This requires an inter-relationship between the Functional and the Physical architecture views as shown by the bold two-way arrow. It is critical that the granularity of the two architecture views be comparable and that partitions have taken place in the hierarchical decompositions in a manner that allows functions or activities to be

assigned unambiguously to resources and vice versa. Once the parameter values and properties of the physical systems have become part of the data base of the executable model, performance evaluation can take place.

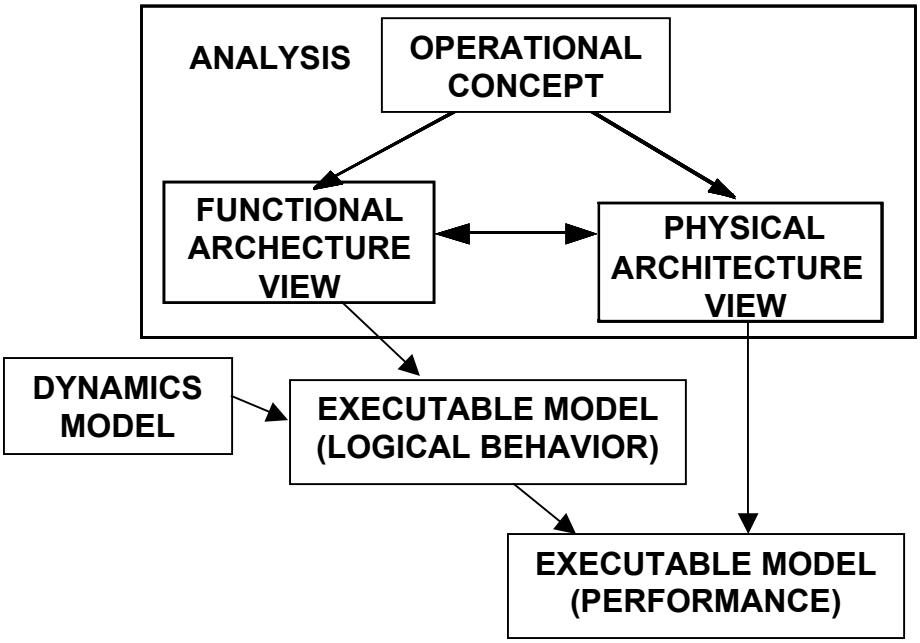

Figure 15. The Synthesis Phase

3.8 Logical, Behavior, and Performance Evaluation

Measures of Performance (MOPs) are obtained either analytically or by executing the model in simulation mode. For example, if deterministic or stochastic time delays are associated with the various activities, it is possible to compute the overall delay or to obtain it through simulation. Depending on the questions to be answered, realistic scenarios of inputs need to be defined that are consistent with the Operational Concept. This phase allows for functional and performance requirements to be validated, if the results obtained from the simulations show that the Measures of Performance are within the required range. If not, the systems may need to be modified to address the issues that account for the encountered problems.

This is actually an iterative process, as shown in Fig. 16. The Executable model can be used both at the logical and behavioral level as well as the performance level. The latter requires the

inclusion of the Physical architecture. In one consistent architectural framework supported by a set of models, requirements analysis, design, and evaluation can be performed. The creation of the Executable Model allows the focus of discourse to be on the behavioral and performance aspects of the architecture. Furthermore, the process provides a documented set of models that collectively contain all the necessary information.

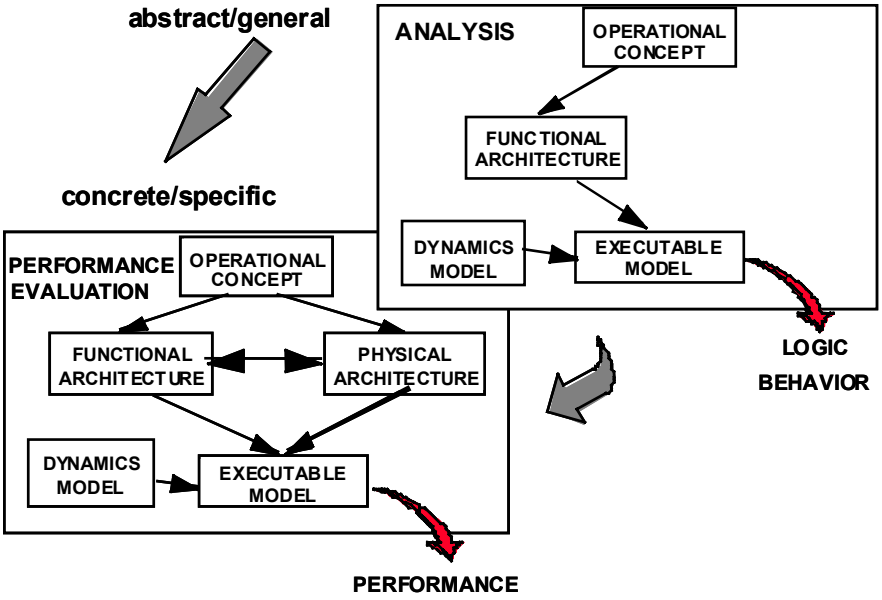

Figure 16. An iterative process

4. C4ISR ARCHITECTURE FRAMEWORK, VERSION 2 PRODUCTS.

Having reviewed the traditional systems engineering process from creating system architectures, we now turn our attention to the C4ISR Framework Version 2. We start with a brief review of the products that are specified by the Framework. This will be followed by the description of a process for generating these products after the architecture has been created using the systems engineering process.

The C4ISR Framework presents four views: These are All View, Operational Architecture View, Systems Architecture View, and the Technical Architecture View. Each view describes a particular characterization of the architecture using a set of products that are graphical, tabular, or textual. The Framework characterizes each product as either essential or supporting. This is an unfortunate terminology because it has the connotation that the supporting views are

secondary. As we shall see, this is not the case. Most of the supporting products are necessary to produce a consistent and coherent description of an architecture and validate the essential products. The intent of the characterization is to identify those products that are the minimum essential for presenting or delivering an architecture to the DoD. It is not intended to mean that the supporting products are not necessary for describing an architecture. The products are listed in Tables 1 through 4.

Table 1. All View Products

Applicable Architecture View	Product Reference	Architecture Product	Essential or Supporting
All Views (Context)	AV-1	***Overview and Summary Information***	**Essential**
All Views (Terms)	AV-2	***Integrated Dictionary***	**Essential**

For completeness, the next section provides a brief description of each of the products of the Framework.

4.1 Architecture Views and Products

As shown in Table 1, the All View is comprised on two products, both essential. The Overview and Summary Information Product, AV-1, contains summary textual information that will allow quick reference and comparison among architectures. This information includes the name of the architecture and the architect, its purpose, scope, and context. It also describes major findings and recommendation that are based on the architecture. The Integrated Dictionary is AV-2. At a minimum, it is a glossary with definitions of terms used in every product in the architecture description. The Integrated Dictionary is essential for allowing the architecture to stand alone, without reference to any other documents.

Table 2. Operational Architecture View Products

Operational	OV-1	***High-level Operational Concept Graphic***	**Essential**
Operational	OV-2	***Operational Node Connectivity Description***	**Essential**
Operational	OV-3	***Operational Information Exchange Matrix***	**Essential**
Operational	OV-4	***Command Relationships Chart***	Supporting
Operational	OV-5	***Activity Model***	Supporting
Operational	OV-6a	***Operational Rules Model***	Supporting
Operational	OV-6b	***Operational State Transition Description***	Supporting
Operational	OV-6c	***Operational Event/Trace Description***	Supporting
Operational	OV-7	***Logical Data Model***	Supporting

The *Operational Architecture* view is a description of the tasks and activities, operational elements, and information flows required to accomplish or support a military operation. It is composed of seven products. OV-1, the High Level Operational Concept Graphic is the most flexible of all the products designed to convey a high level description of the operation that the architecture supports. It is generally composed of nodes, in the form of icons, and connectors. The icons represent organizations, assets, missions or tasks, and the connectors show information flows or connectivity. The graphic can also show the relative geographic position of assets and tasks. OV-1 must be accompanied with a textual description of the operational concept depicted in the graphic, even though this is not specified in the Framework document.

OV-2, is the Operational Node Connectivity Description. This essential product is a directed graph, whose nodes are operation nodes or elements and whose arcs, called need lines, show necessary connectivity and the flow of operational information elements between the nodes.

Each node is annotated with the activities it performs and each need line is annotated with the operational information element that flows from one operational node to another.

OV-3 is the Operational Information Exchange Matrix, an essential product. It contains, in tabular form, information about each Operational Information element that is contained in Operational Node Connectivity Description. For each element it lists the producing and consuming operational node and activity as well as general information including a description, size, composition, frequency of occurrence, timeliness requirements, throughput, security level, and interoperability requirements.

OV-4 is the Command Relationship Chart. This supporting product describes a key organizational aspect of the operational concept that the architecture supports. In its most basic form it is the standard organizational chart common to all DoD organizations. Its intent is to illustrate important relationships between organizational elements in the architecture. Such relationships include command, control, and coordination. OV-4 shows an operational perspective of fundamental roles and management relationships between entities in the architecture. These relationships typically form the basis of some of the connectivity requirements in the architecture.

OV-5 is the Activity Model. In its illustrations of the activity model, the Framework uses the Integrated Definition 0 (IDEF0) as the modeling technique. IDEF0 has been standardized in Federal Information Processing Standard (FIPS) Publication – 183. It is a hierarchical structure of activities, represented by boxes, and data or information exchanged between activities shown as arrows between the boxes. The Framework states that these arrows include inputs, controls, outputs, and mechanisms (ICOMs) that is standard in the IDEF0 formalism. The Framework does not mandate IDEF0 for the activity model and states that other modeling techniques may be used. This is important because it allows some of the techniques used in object orientation to be used for OV-5.

Table 3. System Architecture View Products

Systems	SV-1	***System Interface Description***	**Essential**
Systems	SV-2	***Systems Communications Description***	Supporting
Systems	SV-3	***Systems² Matrix***	Supporting
Systems	SV-4	***Systems Functionality Description***	Supporting
Systems	SV-5	***Operational Activity to System Function Traceability Matrix***	Supporting
Systems	SV-6	***System Information Exchange Matrix***	Supporting
Systems	SV-7	***System Performance Parameters Matrix***	Supporting
Systems	SV-8	***System Evolution Description***	Supporting
Systems	SV-9	***System Technology Forecast***	Supporting
Systems	SV-10a	***Systems Rules Model***	Supporting
Systems	SV- 10b	***Systems State Transition Description***	Supporting
Systems	SV -10c	***Systems Event/Trace Description***	Supporting
Systems	SV-11	***Physical Data Model***	Supporting

Table 4. Technical Architecture View Products

Technical	TV-1	***Technical Architecture Profile***	**Essential**
Technical	TV-2	***Standards Technology Forecast***	Supporting

OV-6 is called the Operational Activity Sequence and Timing Descriptions. It is composed of three supporting products, the Operational Rule Model (OV-6a), the Operational State Transition Description (OV-6b), and the Operational Event/Trace Description (OV-6c). These set of products are used to describe the architecture's dynamic properties. The Operational Rule Model captures business requirements and concept of operation information. The rules must be consistent with the activity and logical data models (OV-7) as well as the State Transition Diagrams (OV-6b). The rules are expressed in a formal language, e.g. Structured English, Decision Trees and Tables, or Mathematical Logic. The Operational State Transition Description (OV-6b) models the architecture's response to specific stimuli. State Transition Diagrams are the modeling technique that is used. The Operational Event/Trace Description (OV-6c) trace sequence of events between operational nodes. In general they have not been used in Structured Analysis approaches, however they are one of the standard models of object orientation.

OV-7 is the Logical Data Model (LDM). It is used to describe the data requirements that comprise the operational information exchange elements. It shows data entities with their attributes and relationships between those entities. The Framework does not specify a specific modeling technique for the LDM, leaving the select up to the architect depending on the purpose of the architecture. The Framework illustrates the data model using the FIPS - 184 IDEF1X as an example of a formal data model.

The System Architecture View is a description, including graphics, of systems and interconnections providing for, or supporting, warfighting functions. It is composed of one essential product and twelve supporting products. SV-1, the System Interface Description, is the only Essential Product in the System Architecture View. It is similar to the physical architecture of the Structured Analysis approach. It is a graphic consisting of nodes and connecting links. The nodes are called system nodes and should map to the operational nodes of the operational node connectivity description, OV-2. The system nodes represent the physical implementation of one or more operational nodes. The System Interface Description identifies interfaces between system nodes, systems within system nodes, and system components. The interfaces are in the form of communications links or paths. The System Interface Description has four variants. The architect selects the variant(s) needed to support his architectural effort. These

variants are the Internodal Perspective - Node Edge to Node Edge interfaces, the Internodal Perspective - System-to-System Interfaces, the intranodal perspective that shows the interfaces between systems within a system node, and the intrasystem perspective, that shows the interfaces between the components that comprise a system within a system node.

SV-2 is the Systems Communication Description. It provides more detail of the system interfaces. It focuses on the physical implementation aspects of the needlines in the Operational Node Connectivity Description and also depicts pertinent information about the communications elements and services. It has two variants, an internodal perspective and an intranodal perspective.

SV-3 is Systems^2 Matrix. It is similar to an N2 matrix where the systems are listed in both rows and columns and each cell provides a description of the interface between the systems is one exists. Characteristics of the interface include status (existing, planned, etc), category (C2, intelligence, etc.), classification level and means (specific network).

SV-4 is the Systems Functionality Description. It is an activity model based on the notion of Data Flow Diagrams (DFDs). The activities, called transformations in DFDs, represent the system functions that are performing the operational activities described in the Activity Model of the Operational Architecture View (OV-5). Its purpose is to depict data flows and data stores within system perspective.

SV-5 is the Operational Activity to System Function Traceability Matrix. It is a link between the Operational and System Architecture Views. It reflects the result of the allocation the activities in the Operational Architecture View to the system functions in the System Architecture View.

SV-6 is the System Information Exchange Matrix. It is similar to and complements the Operational Information Exchange Matrix. It is tabular in form and describes the information exchanges between systems within a node and from systems to systems between nodes. Associated with each system information element is a system function that either produces or receives the system information element. The focus of this matrix is to describe the implementation of the information exchanged between systems (and their functions). Therefore the matrix contains descriptions of characteristics of each element such as content, media, format, security level, frequency of exchange, timeliness requirements, etc.

SV-7 is the System Performance Parameters Matrix. This product provides current and predicted or required future performance characteristics for each system component and element in the system architecture view. This information is essential for the executable model if Performance Evaluation is to be accomplished.

SV-8 is the System Evolution Description. It is a timeline type of depiction of plans for modernizing or evolving the system over time.

SV-9 is the System Technology Forecast. It is a detailed description of emerging technologies that can impact the architecture. It contains predictions about the availability of the technology and the potential impact the technology can have on the architecture.

SV-10a, 10b, and 10c comprise the System Activity Sequence and Timing Descriptions. They were intended to describe the dynamic behavior of the architecture. Recent discussions between the DoD and the authors reveals that the System Rules Model (SV-10a) and the System State Transition Description (SV-10b) are no longer included as products in the System architecture View. The Systems Event/Trace Description (SV-10c) is a construct used in Object-Oriented methodologies. Like its counterpart in the Operational Architecture View, it is intended to shown the sequence of messages that are sent between System Nodes for a given initial situation. Thus they present a static view of dynamic behavior.

SV-11 is the Physical Data Model. It is used to describe how the information represented in the Logical Data Model (OV-7) is implemented in the Systems Architecture View. There is a mapping between the Logical Data Model and the Physical Data Model. There is considerable flexibility as to the form that the Physical Data Model may take. Typical descriptions include message formats, file structure descriptions and physical data schema representations.

The Technical Architecture View conveys the set of rules that governs system implementation. As was discussed before, it is analogous to the building code that an architect invokes in creating the architecture for a house or building. This view is composed of two products, one essential and the other supporting. The Technical Architecture Profile (TV-1) is the essential product. This product references the technical standards that apply to the architecture and how they need to be implemented. A typical Technical Architecture Profile is a table that list Service Areas (e.g. operating system, Data management, etc.), the Service (Kernel, Data Interchange, etc.) and the Standard to be applied (FIPS Pub 151-1, FIPS Pub 127-2, etc.).

TV-2 is the Standards Technology Forecast. It extends the information contained in TV-1 to list anticipated updates and changes in applicable standards for the architecture.

5 DEVELOPING A PROCESS FOR CREATING THE FRAMEWORK PRODUCTS

The C4ISR Framework Version 2.0 products represent a different perspective than that of the traditional Structured Analysis that has orthogonal Functional and Physical Architecture views. Traditional tools and techniques support the Structured Analysis approach and research has shown [Levis et al., 99] that this approach is complete only in the sense that it contains all the information and data needed to produce an executable model. Actually, the Structured Analysis approach underlies the Framework products. Several of the Framework products are identical to those used in Structured Analysis.

The only guidance provided by the Framework document is shown in Fig. 17. Six steps are defined for the development of an architecture. These are fundamental steps; it is the responsibility of the (chief) architect that they be followed in every architecture development effort. The first step reinforces the idea that architectures must be designed for a purpose, to address a particular set of problems. This purpose and the associated problems must be articulated before the development of the architecture and the production of products begins. The next step is the determination of the scope of the architecture. In systems engineering terms, this corresponds to determining the system boundary, i.e., determining which elements are going to be considered within the architecture and which will be considered as part of the environment. The third step is closely related to the first one: the choice of attributes that are to be included is directly dependent on the questions to be answered. At this point, the architect is ready to determine which supporting products are needed in addition to the essential products. Note that the first four stages require the involvement of few persons – the architect, the users, and a small staff that supports the architect. The fifth step is the labor intensive one in which the architecture is designed and the products developed. This step is guided by the previous four steps; failure to do that may easily result in a set of products that will be unable to address the last, and most critical step, the use of the architecture for the intended purpose, i.e., to provide answers to the original questions. The development of the executable model could be included either in step 5 or 6. Note that the first four steps are really the responsibility of the DoD organization that is

developing the C4ISR architecture while the 5th and 6th steps can be done by industry; the first fours steps set up the requirements while the last two are the actual design.

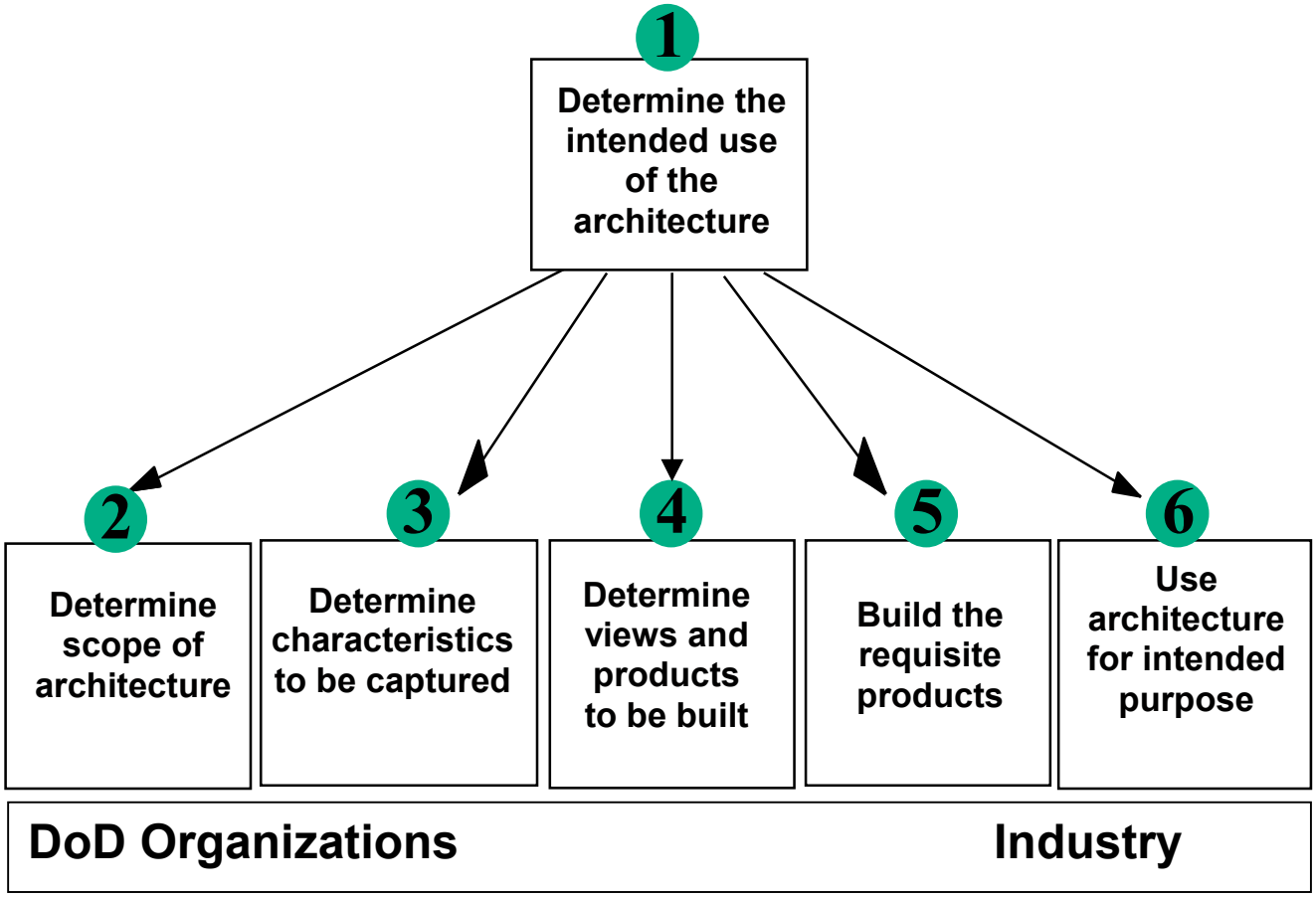

Figure 17 Universal guidance for Architecture development

The steps addressed by this and the two companion papers [Wagenhals et al., 00], [Bienvenu et al., 00] are step 5 and part of step 6. The two steps can be decomposed as shown in Fig. 18. First, the architect and his team develop the architecture using the Structured Analysis tools and techniques that were described in Section 3, or any other approach such as object orientation. If the architecture representation is complete in the sense that the executable model is obtainable, then the C4ISR Architecture Framework products can be derived. This statement requires clarification. It is possible to derive an executable model from the Operational Architecture view alone. Such a model can only address some logical and behavioral questions. To address

performance questions, the System Architecture view is needed. If the architecture has been defined and all the information resides in an integrated data base, then one can see the products as nothing else but a set of reports drawn from the data base. Indeed, this is the concept that allows the use of alternative methodologies for the design of the architecture and forms the theoretical justification for a software product such as JCAPS [00].

The executable model requires some additional information not specified explicitly in the Framework: it needs timing information and it needs a set of scenaria to run simulations and collect data for the Measures of Performance and the Measures of Effectiveness.

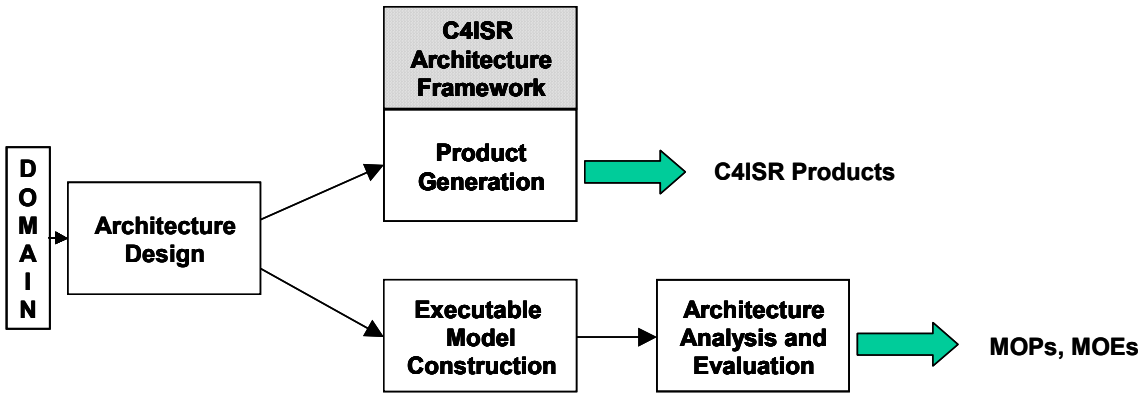

Figure 18. Template of the Architecture design process

Given this design process, the next step is to develop specific procedures that realize the functions indicated in each one of the boxes. Two different procedures have been developed, one based on Structured Analysis [Wagenhals et al., 00] and one on object orientation [Bienvenu et al., 00]. The process that follows the lower branch of Fig. 18 is straightforward and well understood for a number of years [Levis, 99]. However, one may think of the upper branch as a constraint on the lower one: it is not sufficient to obtain any design and any executable model; the design should produce the products specified by the C4ISR Architecture Framework. To achieve that, a reverse engineering procedure was followed.

Each one of the Framework products was considered a data entity composed of other data entities and a formal IDEF1x data model of the set products was created. This model contained over 100 entities. Of those 100 entities, 16 were identified a key entities that are necessary to

understand the mapping between the Structured Analysis products and the Framework products. These entities and their relationships are shown in Fig. 19 in the form of an Entity-Relationship diagram.

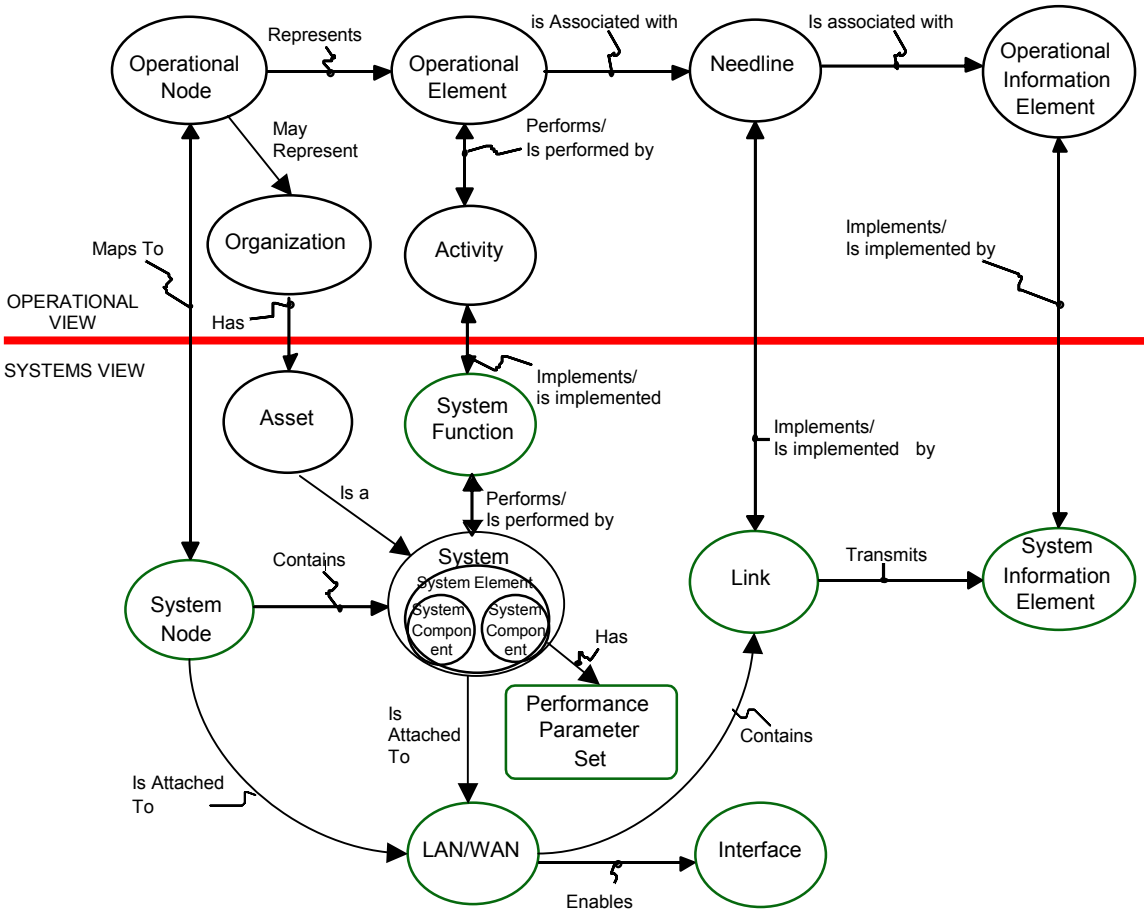

Figure 19 Key Entities

In this E-R diagram, the entities are partitioned into two categories: entities associated with the operational architecture view and entities associated with the systems architecture view. This partition is indicated by the horizontal line in Fig.19.

Starting with the operational view, the key entities are the *operational nodes* and *operational element*s. The operational nodes are the graphic constructs that appear in the Operational Node Connectivity Description (OV-2). The operational elements are specified in the Operational Information Exchange Matrix (OV-3). Each graphical operational node in OV-2 represents one or more operational elements, and, therefore, can contain one or more operational elements.

These nodes and the elements they represent do not have to be real physical facilities. Instead they can represent notional or "virtual" nodes and elements based on the operators' view of operational roles. These roles are normally associated with *organizations* that have *assets* that comprise the *systems* that ultimately perform the *activities* of the architecture. Thus each operational node represents or contains operational elements that perform activities and receive and produce *operation information elements*. The Operational Node Connectivity Description also has directed arcs between the operational nodes that are called *needlines*. The needlines indicate the need for connectivity between operational nodes. The need for the connectivity is based on the operational information elements that must flow between the activities that are performed at the operational nodes.

There are several mappings between key entities of the Operational Architecture View and Systems Architecture View. The operational nodes map to systems nodes that are graphical constructs in the System Interface Description (SV-1) and the Systems Communications Description (SV-2). Each system node represents or contains systems. Systems, in turn are composed of system components that are in turn composed of system elements. These are also part of SV-1 and SV-2. These systems, components and elements perform system functions that are the physical implementation of the activities described in the operational architecture view. Because they represent real physical entities, each system, component, and element has performance parameters associated with it. The systems, components and elements are connected together via communications assets such as Local Area Networks (LANs) and Wide Area Networks (WANs). These networks provide interfaces between the systems, components, and elements. These networks provide communications links between the systems that implement the needlines of the Operational Architecture View. System information elements, which are the physical implementation of the operational information elements flow over the links between the systems, component, and elements as the system functions are performed.

Together, these key entities support the creation of most of the Framework products. For the operational architecture view, in addition to OV-2 and OV-3, the organizations are represented in the Command Relationship Chart (OV-4), the activities and their relationships are specified in the Activity Model (OV-5), and the Operational Information Elements are described in the Logical Data Model (OV-7). For the Systems Architecture View, in addition to SV-1 and SV-2, the system functions and their relationships are expressed in the System Functionality

Description (SV-4), the relationship between system functions are described in the System^2 Matrix (SV-3), the allocation of the activities to system functions is defined in the Operational Activity to System Functions Traceability Matrix (SV-4), the characteristics of the system information elements are described in the Physical Data Model (SV-11), the relationship between system information elements and system functions are specified in the System Information Exchange Matrix (SV-6), and the system performance parameters are described in the System Performance Parameter Matrix (SV-7).

Given the understanding of the key entities of the Framework products and the formal procedure for creating an architecture using Structured Analysis, it is possible to show the relationship between the elements of the Structured Analysis descriptions of the architecture and the Framework products. Recall that the structure analysis process begins with the description of the operational concept for accomplishing a mission that is used to guide the development of both the functional architecture and the physical architecture views. The latter also includes organizational models. Throughout the Structured Analysis process, the functional and physical architecture views are balanced. This process was depicted in Fig. 15 of Section 3.

Figure 20 shows the Analysis phase of Fig. 15 overlaid on the model of the Key Entities. The implied process begins with the creation of the operational concept. Note that this can be defined in the Operational Concept Graphic (OV-1). The operational concept guides the development of the functional decomposition, the physical architecture composed of system nodes and links, and the organization model. It also guides the selection of the operational nodes.

The functional decomposition contains the activities and is used to guide the development of the functional architecture. This is composed of the activity model, that is OV-5, the Logical Data Model, OV-7, and the rule model, OV-6a. The dynamics model (OV-6b) is created in the form of a State Transitions Diagram. The arrows between these models reflect the need to ensure concordance between the models. Implied, but not shown is the Integrated Dictionary, AV-2. Figure 20 also depicts the physical architecture view as the system nodes with systems, system elements, and system components, and the links that connect them.

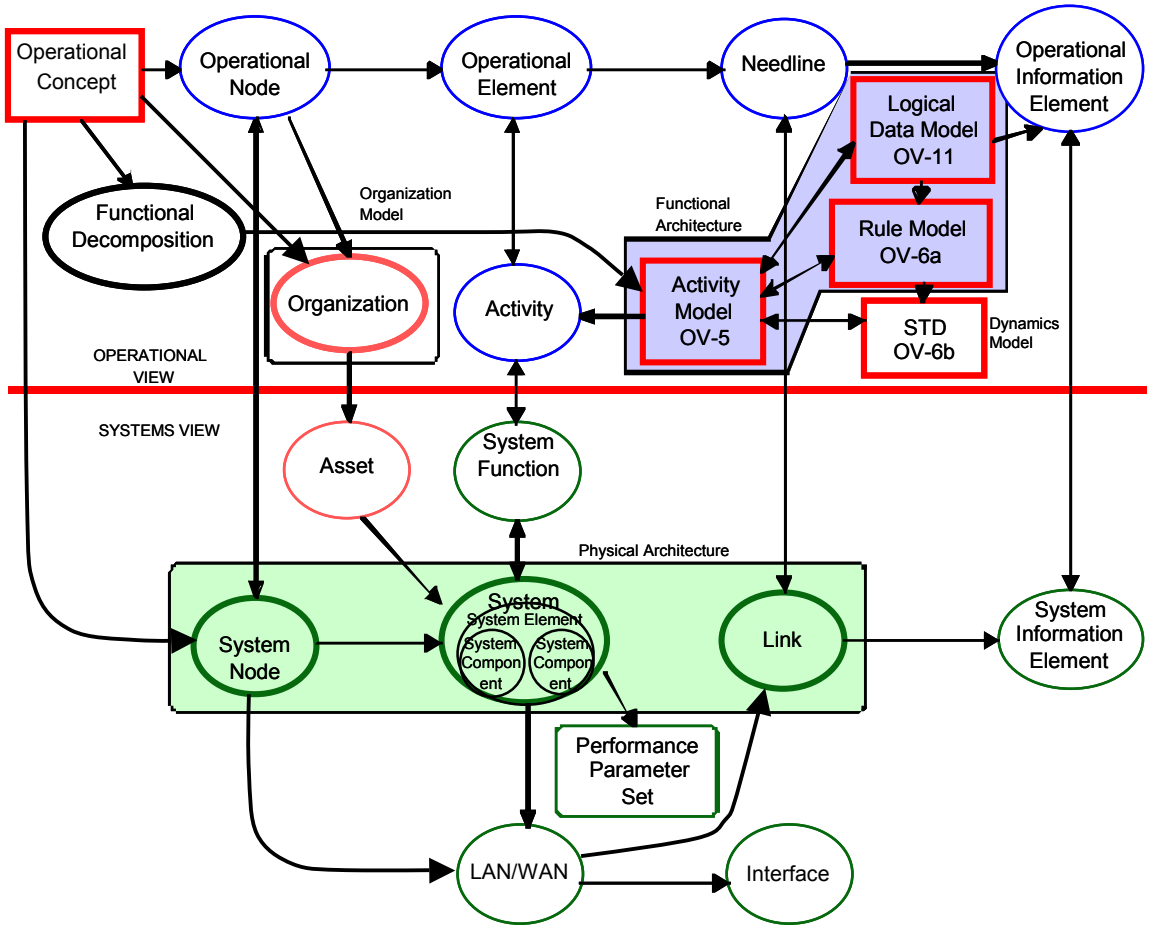

Figure 20. Relationship Between Analysis Phase and Framework Key Entities

These fundamental relationships among the key entities induce a partial ordering of them and of the Framework products. This partial order forms the basis for a process for creating the architecture. The process can be based either on Structured Analysis constructs, as is done here, or on Object-Oriented ones [Bienvenu et al., 00]

Note that this is not presented as a recommended process to use in developing a C4ISR architecture. Rather, it is a reverse-engineered process, based on Structured Analysis constructs, that identifies the interrelationships among products. Organizations involved in the design of architectures often have their own approach or process for developing architectures but are uncertain whether their process will meet the C4ISR Architecture Framework requirements. We have been suggesting to these organizations (both military and industrial) that they compare their process to the one described here as a means of assessing whether they have covered all the aspects they need and whether they need to modify or enhance their process by recognizing some

interrelationships between products at an early stage. For example, this process identifies Information Exchange Requirements, an essential product, as one of the outputs of the process, not one of the inputs. This is not to say that such data need not be collected – rather, the final list is the result of the analysis embedded in the execution of any thorough process.

7. CONCLUSIONS

We have discussed the creation of architectures for information systems in general and explored a process for creating the Essential and Supporting Products of the DoD C4ISR Architecture Framework Version 2.0. The Framework, by its depictions of example products, has a Structured Analysis bias in its representation of the products that represent three views of an architecture. It does not provide or recommend a process for creating these products, but only some universal guidance. In this paper, we have shown using Structured Analysis that it is possible to develop a process that generates these products and also provides the necessary information for the derivation of an executable model. In the two companion papers, explicit processes are described and the same example is used to show how these processes can be implemented.

References

Bienvenu, P. P., D. Kim, and A. H. Levis, C4ISR Architectures III: An Object-Oriented approach to architecture design, C3I Center, George Mason University, Fairfax, VA, April 2000.

C4ISR Architecture Framework Version 2.0. C4ISR Architecture Working Group. Department of Defense, December 18, 1997.

DeMarco, T., Structured Analysis and systems specification, Prentice Hall, Englewood Cliffs, NJ, 1979

FIPS 183: Federal Information Processing Standard # 183: Integration definition for function modeling, National Institute for Standards and Technology, Gaithersbourg, MD

FIPS 184, Federal Information Processing Standard # 184: Integration definition for information modeling, National Institute for Standards and Technology, Gaithersbourg, MD

Gane, C.,and T. Sarson, Structured systems analysis: Tools and techniques, Prentice Hall, Englewood Cliffs, NJ, 1978

Jensen, K., Coloured Petri nets, Springer-Verlag, Berlin, 1992.

JCAPS: Joint C4ISR Architecture Planning/Analysis System, Status and Future Directions briefing, OASD(C3I), April 2000.

Levis, A. H., "Systems Architectures," in Handbook of Systems engineering and Management, A. P. Sage and W. B. Rouse (Editors), Wiley, NY, 1999, pp. 427-456.

Levis, A. H. and L. W. Wagenhals, Architecting Information Systems, GMU/C3I-165-R, C3I Center, George Mason University, Fairfax, VA 1999; also at http://viking.gmu.edu/http/syst621/Syst621.htm

Marca, D.A., and C. L. McGowan, Structured Analysis and design technique, McGraw-Hill, New York, 1987

McLeod, Jr., R., Systems analysis and design, Dryden, Fort Worth, TX, 1994.

Rechtin, E., Systems Architecting: Creating & Building Complex Systems. Prentice Hall, Englewood Cliffs, NJ, 1991

Rechtin, E., The art of systems architecting. IEEE Spectrum, Oct. 92, pp. 66-69.

Rechtin, E., and M. Maier, The art of systems architecting. CRC Press, Inc., Boca Raton, FL, 1996

Rumbaugh, J., M. Blaha, W. Premerlani, F, Eddy, and W. Lorensen, Object-oriented modeling and design, Prentice Hall, Englewood Cliffs, NJ,1991

Sanders, G. L., Data modeling, Boyd & Fraser, Danvers, MA, 1995

Solvberg, A., and D. C. Kung, Information systems engineering, Springer-Verlag, Berlin 1993

Wagenhals, L. W., I. Shin, D. Kim, and A. H. Levis, C4ISR Architectures II: A Structured Analysis approach to architecture design, C3I Center, George Mason University, Fairfax, VA, April 2000.

Ward, P., and S. Mellor, Structured development of real-time systems, Yourdon Press, New York, 1986

Yourdon, E., Modern Structured Analysis, Yourdon Press, Englewood Cliffs, NJ, 1989

Yourdon, E., & Constantine, L. (1975). Structured design. New York: Yourdon Press.